Panoramic Image Mosaicing

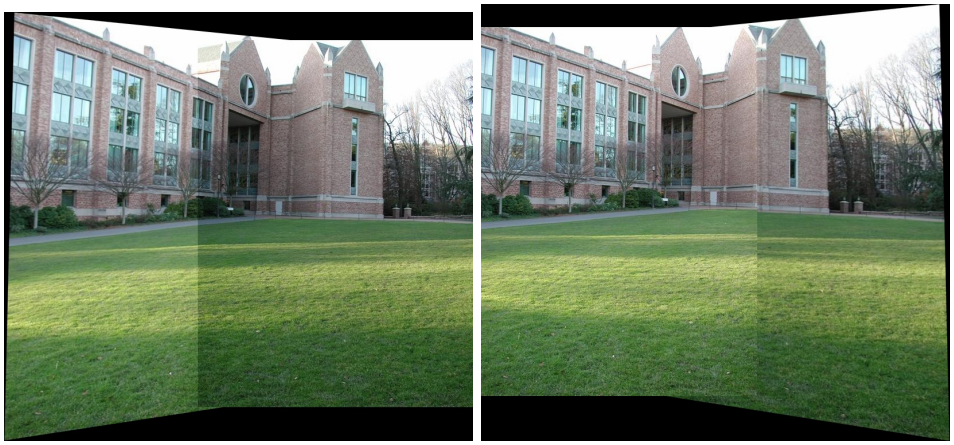
Observations and Results

1 Auto Mosaicing

The goal here is to automatically detect correspondent points in input images and mosaic them. The method is described in section 1.2.

1.1 Results

(a) I1 (b) I2



(c) I1 as reference (d) I2 as reference

Figure 1: Campus Auto Mosaicing

(a) I1 (b) I2

(c) I1 as reference (d) I2 as reference

Figure 2: Yard Auto Mosaicing

(a) I1 (b) I2

(c) I1 as reference (d) I2 as reference

Figure 3: Ledge Auto Mosaicing

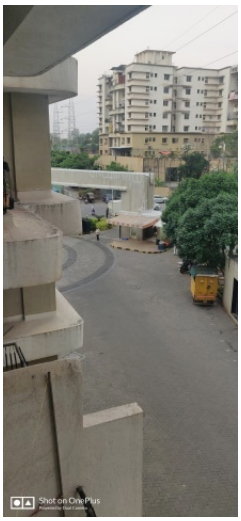
(a) I1

(b) I2

(c) I1 as reference

(d) I2 as reference

Figure 4: Society Auto Mosaicing



(a) I1

(b) I2

(c) I1 as reference

(d) I2 as reference

Figure 5: Gate Auto Mosaicing. This example has an interesting quirk as described in Q.1.3 in section 1.2

1.2 Questions

Q.1.1: Did you filter? If yes, then how did you filter? How many point correspondences did you find before and after filtering?

We found the point correspondences using the Brute Force Matcher provided by OpenCV, `cv2.BFMatcher`. For each descriptor in the first set, this matcher naively finds the closest descriptor in the second set by trying each one [3]. ORB provides binary descriptors, where the descriptors are bitstrings and not numbers [4]. Thus we had to use the Hamming Norm. We enabled the `crossCheck` parameter to filter consistent pairs of correspondences.

We sorted based on the Hamming distances (metric similar to edit distance for bitstrings) and chose the top 20 matches. Even the top matches are susceptible to mis-identifying similar-looking but unrelated points to be correspondent points. These matches are treated as outliers. In order to deal with these, we used the RANSAC (Random Sample Consensus) method while constructing the homography matrix with the reprojection error threshold set as 4 [2].

Before filtering, we obtained an average of 150-200 matches. We filtered the top 20 and further subjected them to outlier elimination.

Q.1.2: How did you choose the values for parameters in the process of finding 'good' point-correspondences?

For filtering the sorted list of matches, we first chose all the points. This gave us frequent bad results. Our intuition is that the RANSAC outlier elimination fails to reach a consensus in the presence of several bad matches.

We chose to filter top 10 points but ran into occasional mis-behavior. Our intuition is that if the top matches are concentrated in a small patch in both the images, it fails to adapt to the perspective change in the remaining part of the overlap.

Filtering the top 20 points worked the best for us.

For choosing the RANSAC reprojection error threshold, we used the heuristic [2] that it should lie in the range 1 and 10. We explored a couple of files and `ransacReprojThreshold`=4 worked well for us.

Q.1.3: Did the dataset you used earlier work for the manual case work as well? Explain.

The automatic algorithm worked for the Figure 4 (Society) despite the "OnePlus" watermark at the lower left (possibly due to the tree background underplaying its resemblance).

It failed for the Figure 5 (Gate). Here all the top correspondences seem to have been between elements of the "OnePlus" logo. Thus the reference image exactly overlays the transform image in both cases (c) and (d).

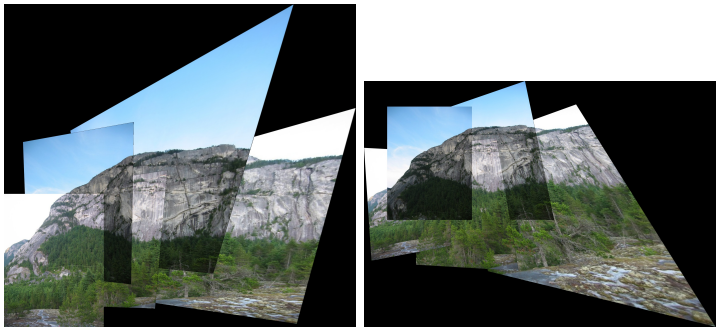
2 Generalized Mosaicing

In this section, we generalize to stitching together 'n' input images, given also the desired reference image. The method is described in section 2.2.

2.1 Results

2.1.1 Mountain

(a) (b) (c) (d) (e)



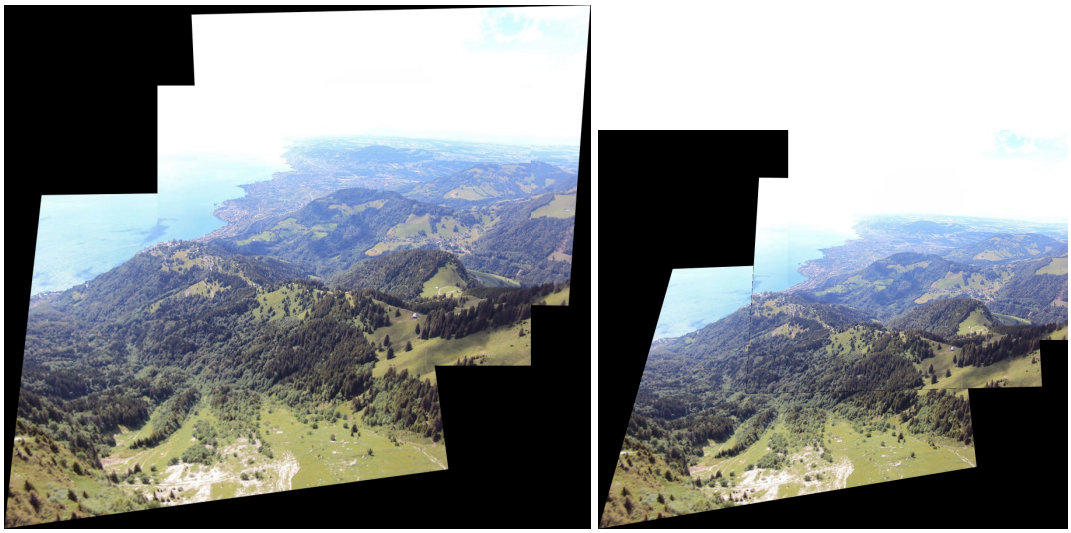
(f) Reference Image: I1 (g) Reference Image: I2

(h) Reference Image: I3 (i) Reference Image: I4

Figure 6: Mountain Generalized Mosaicing. Reference image I3 gives best result. Program crashes on taking reference image I5.

2.1.2 Ledge

(a) (b) (c)



(d) Reference Image: I1 (e) Reference Image: I2

Figure 7: Ledge Generalized Mosaicing. Reference image I1 gives best result. Program crashes on taking reference image I3.

2.1.3 Yard

(a) (b) (c) (d) (e)

(f) Reference Image: I2 (g) Reference Image: I3 (h) Reference Image: I4

Figure 8: Yard Generalized Mosaicing. Reference image I3 gives best result. Program crashes on taking reference image I1.

2.1.4 Room

(a) (b) (c)

(d) Reference Image: I1 (e) Reference Image: I2 (f) Reference Image: I3

Figure 9: Room Generalized Mosaicing. Reference image I2 gives best result.

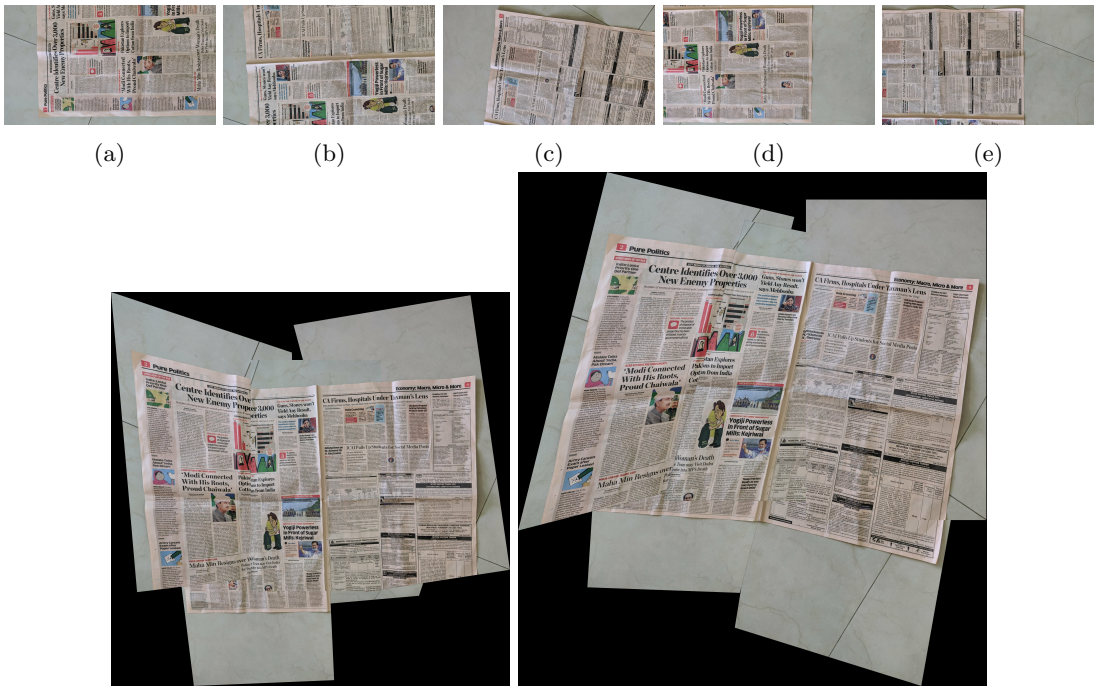
2.1.5 Yosemite

(a) (b) (c) (d)

(e) Reference Image: I1 (f) Reference Image: I2 (g) Reference Image: I3

Figure 10: Yosemite Generalized Mosaicing. Reference image I2 gives best result. Program crashes on taking reference image I4.

2.1.6 Newspaper



(a) (b) (c) (d) (e)

(f) Reference Image: I2 (g) Reference Image: I3

Figure 11: Newspaper Generalized Mosaicing. Reference image I3 gives best result. Program crashes on taking reference image I1, I4, I5.

2.2 Questions

Q.2.1: Explain the method you followed. Do you get the result if you choose any image to be the reference image? If no, what issues are you facing? Give your opinion on reason behind these issues.

We iteratively run the `stitchPairOfImages()` routine defined in the `helper.py` script. We position the reference image at the center of a large blank canvas and stitch an adjacent transform image in the sequence. This mosaic image becomes the reference image for the next iteration of `stitchPairOfImages()` and we update the transform image to the next adjacent image in the sequence. (More specifics illustrated in comments in `helper.py`). This ensures that the reference image is undistorted and un-overwritten, thus staying intact.

We use a masking technique to get rid of black grooves while overlaying reference image on transformed image [6].

No, we do not get appealing results occasionally on choosing the extreme indexes as reference images. This owes to error propagation in the perspective transform. If one transform image has moderate skew error, the next adjacent transform image gets heavily skewed. The program crashes as either the demands for the blank canvas dimensions exceeds the system's capacity or the perspective transform matrix itself runs into a singularity.

Q.2.2: Here, we assumed that you know the sequence in which you have to stitch the images. How would you modify your approach if you don't know the sequence? Describe the approach briefly.

A brute force technique can be used to know the best sequence for stitching the images. We can run for $\binom{n}{2}$ (where n is the number of images to be mosaiced) combinations of pair of images and check whether `cv2.BFMatcher` gives good mappings of feature points between 2 images and set `crossCheck` param to be true. This will ensure that the two features in both images match each other. Further we will check the best matching Hamming distance value and set a threshold to consider this pair of images having any point correspondence. If the distance is very high, then we will safely assume that pair of images dont have any point correspondence. Once we know all the possible pairs of images which have any overlap or point correspondences based on above process, we will just take the best (n-1) pairs from this set (based on Hamming distances of best matchers points) to cover all images and then run our algo to make our final mosaiced image.

Q.2.3: How does the functionality you are providing (or could provide) differ from the Stitcher class.

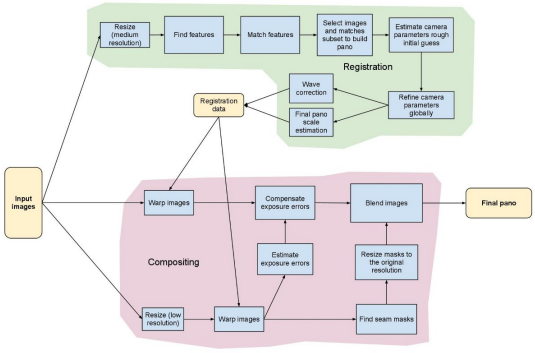


Figure 12: Stitcher class pipeline [7]

The differences between our functionalities from the Stitcher class are as follows:

1. The class does a preliminary resize to medium resolution to find and match features. We have directly worked with the input image dimensions.

2. Their approach involves multi-step estimation of camera parameters going from a rough initial guess and refining it in steps.

3. They rescale the input images to low resolution to obtain seam masks which they use to blend the stitched images. We have suggested a rudimentary Gaussian blur for the same.

Q.2.4: The mosaicing results show a "seam". What techniques can be used to remove the seam?

The seam between 2 images mainly come due to the different lighting conditions between 2 images. For removing this, we can use the Histogram matching technqiue to match the histograms of 2 images and hence equalise the lighting conditions. This can be followed by applying a gaussian filter on the "seam" line to blur it as we know, gaussian filter acts as a Blurring filter.

References

[1] https://docs.opencv.org/master/

[2] https://docs.opencv.org/master/d9/d0c/group__calib3d.html#ga4abc2ece9fab9398f2e560d53c8c9780

[3] https://docs.opencv.org/4.5.1/d3/da1/classcv_1_1BFMatcher.html#ac6418c6f87e0e12a88979ea57980c020

[4] https://answers.opencv.org/question/147525/which-norm-is-the-best-to-match-descriptors/

[5] https://stackoverflow.com/questions/13538748/crop-black-edges-with-opencv

[6] https://stackoverflow.com/questions/2169605/use-numpy-to-mask-an-image-with-a-pattern

[7] https://docs.opencv.org/3.4/d1/d46/group__stitching.html